Theoretical Computer Science 438 (2012) 1–12

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# The derivational complexity of string rewriting systems

Yuji Kobayashi

*Department of Information Science, Toho University, Funabashi 274–8510, Japan*

ARTICLE INFO

*Article history:*
Received 19 May 2010
Received in revised form 1 February 2012
Accepted 21 February 2012
Communicated by M. Ito

*Keywords:*
Rewriting system
Derivational complexity
Turing machine
Time function
Computable real number

ABSTRACT

We study the derivational complexities of string rewriting systems. We discuss the following fundamental question: which functions can be derivational complexities of terminating finite string rewriting systems? They are recursive, but for any recursive function, there is a derivational complexity larger than it. We relate them to the time functions of Turing machines. In particular, we show that the functions $n^{\alpha}(\alpha > 2)$ and $\alpha^{n}(\alpha > 1)$ for a real number $\alpha$ are equivalent to the derivational complexity of some finite string rewriting systems if the computational complexity of $\alpha$ is relatively low (for example, $\alpha$ is rational, algebraic, $\pi$ or $e$). On the other hand, they cannot be equivalent to any derivational complexities if the complexity of $\alpha$ is high.



## 1. Introduction

Term rewriting is a simple but powerful abstract model of computation on which much of the declarative programming is based. When we consider models of computation, one of the fundamental problems is termination. But termination is in general an undecidable property. Since it is essential for verification methods concerning functional and logic programming languages, various termination proof methods have been studied and developed [4].

Once we have established termination of a rewriting system, the next fundamental problem is its complexity. In order to measure the complexity it is natural to look at the maximal length of derivation sequences as suggested by Hofbauer and Lautemann [7]. More precisely, the derivational complexity function relates the length of the longest derivation sequence to the size of the initial term.

In this paper, we treat string rewriting systems. They are special term rewriting systems where all the function symbols are of arity one. Even for these simple systems, the termination problem is undecidable [9]. If a system $R$ is complete (terminating and confluent), the derivational complexity is equal to the complexity of the normal form algorithm of the word problem of $R$ [10].

Many studies have been done about the derivational complexity of term rewriting systems under specific termination techniques (recursive path orders, polynomial interpretations etc., see [11] and the references cited there). In this paper, we discuss the derivational complexity of string rewriting systems under a general situation.

After giving basic notions in the next section, we study a precise relationship between derivational complexities of string rewriting systems and the time functions of Turing machines in Sections 3 and 4. We give a sufficient condition for a recursive function to be equivalent to the derivational complexity of a finite string rewriting system at the end of Section 4. In Section 5, we discuss how to compute the derivational complexity of a given system and give a necessary condition for a function to be equivalent to a derivational complexity. In the final section, we apply these results to the functions $n^{\alpha}(\alpha > 2)$ and $\alpha^{n}(\alpha > 1)$ for a real number $\alpha$, and obtain our main results that they are equivalent to the derivational complexity of some finite string rewriting systems if $\alpha$ is computable in time $C^{2^n}$ for some $C > 1$, but they cannot be equivalent to any derivational complexity if $\alpha$ is not computable in time $C^{C^{2^n}}$ for any $C > 1$.

*E-mail address:* kobayasi@is.sci.toho-u.ac.jp.

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.02.037

We hope that the results obtained in this paper would throw insight into the complexity of general rewriting systems.

## 2. Derivational complexity

Let $\Sigma$ be a (finite) alphabet and let $\Sigma^*$ be the free monoid generated by $\Sigma$. The length of a word $x \in \Sigma^*$ is denoted by $|x|$, and the empty word, the word of length 0, is denoted by 1. Let $\Sigma^n$ be the set of words of length $n$ over $\Sigma$, then $\Sigma^* = \cup_{n\geq 0}\Sigma^n$.

A (string) rewriting system $R$ is a subset of $\Sigma^* \times \Sigma^*$. An element $r = (u, v)$ in $R$ is called a rule of $R$ and written $u \to v$. Suppose that a word $x \in \Sigma^*$ contains $u$ as a subword, that is, $x = x_1 u x_2$ with $x_1, x_2 \in \Sigma^*$, then we can apply the rule $r$ to $x$ and $x$ is rewritten to the word $y = x_1 v x_2$. In this situation we write $x \to_r y$. If there is some rule $r \in R$ such that $x \to_r y$, we write $x \to_R y$, and we call the relation $\to_R$ the *one-step derivation* on $\Sigma^*$ by $R$.

A rewriting system $R$ is *terminating* on $x \in \Sigma^*$ if there is no infinite sequence of derivations

$$x \to_R x_1 \to_R \cdots \to_R x_n \to_R \cdots$$

starting with $x$. $R$ is *terminating* (or *noetherian*), if it is terminating on every $x \in \Sigma^*$.

The maximal length of a derivation sequence starting with $x$ is denoted by $\delta_R(x)$. For $x$ on which $R$ is not terminating, we set $\delta_R(x) = \infty$. The function $d_R : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ defined by

$$d_R(n) = \max\{\, \delta_R(x) \mid x \in \Sigma^n\}$$

for $n \in \mathbb{N}$ is the *derivational complexity* of $R$.

We are interested in what functions can be derivational complexities of terminating finite rewriting systems.

Let $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$. For two functions $f, g : \mathbb{N} \to \mathbb{R}_+ \cup \{\infty\}$, if there is a constant $C > 0$ such that $f(n) \leq C \cdot g(n)$ (resp. $f(n) \geq C \cdot g(n)$) for any sufficiently large $n \in \mathbb{N}$, we write $f = O(g)$ (resp. $f = \Omega(g)$). If both $f = O(g)$ and $f = \Omega(g)$ hold, $f$ and $g$ are called *equivalent*, and written $f = \Theta(g)$.

A function $f : \mathbb{N} \to \mathbb{R}_+ \cup \{\infty\}$ is *super-additive* if

$$f(m + n) \geq f(m) + f(n)$$

holds for any $m, n \in \mathbb{N}$. A super-additive function is non-decreasing. It is easy to see that the derivational complexity of a rewriting system is super-additive.

If there is a derivation sequence of length $\ell$ from $x$ to $y$, we write as $x \to_R^\ell y$. Set

$$\to_R^* = \bigcup_{\ell\geq 0} \to_R^\ell .$$

For an integer $k \geq 1$, a rewriting system $R$ has polynomial (derivational) complexity of degree $k$, if $d_R(n) = \Theta(n^k)$. Any nonempty rewriting system $R$ has at least linear complexity, that is, $d_R(n) = \Omega(n)$.

**Example 2.1.** Let $k \geq 2$ and let $\Sigma_k = \{a_1, a_2, \ldots, a_k\}$. For $2 \leq \ell \leq k$ let

$$C_\ell = \{a_1 a_\ell \to a_\ell a_{\ell-1}, a_2 a_\ell \to a_\ell a_{\ell-2}, \ldots, a_{\ell-1} a_\ell \to a_\ell a_1\},$$

and define a system $P_k$ on $\Sigma_k$ by $P_k = \bigcup_{\ell=2}^k C_k$. Then, $P_k$ has polynomial complexity of degree $k$. In fact, we have a derivation sequence

$$a_1^n a_2^n \cdots a_k^n \to_{P_k}^* a_k^n a_{k-1}^n \cdots a_1^n$$

of length equal to $\Theta(n^k)$, and this is the longest sequence starting with words of length $kn$.

A rewriting system $R$ has *exponential complexity*, if there are constants $C \geq D > 1$ such that

$$D^n \leq d_R(n) \leq C^n$$

for sufficiently large $n \in \mathbb{N}$. The one-rule system $\{ab \to b^2 a\}$ has exponential complexity for example.

Due to [6], a derivational complexity exists in each level of the Grzegorczyk hierarchy of primitive recursive functions. Even the Ackermann's function is attained [7]. Actually, for any recursive function $f$ there is a derivational complexity $g$ such that $g \geq f$ (see Section 4).

## 3. Q-systems and Turing machines

In this paper we only consider deterministic Turing machines. Let

$$M = (\Sigma, Q, q_0, F, \delta)$$

be a $k$-tape Turing machine, where $\Sigma$ is an alphabet (the tape alphabet and the input alphabet are the same), $Q$ is a set of states, $q_0$ is an initial state, $F$ is a set of final states and $\delta$ is a transition function.

Let $\Sigma_b = \Sigma \cup \{b\}$, where $b$ denotes the blank symbol. The transition function $\delta$ is a mapping from $(Q \setminus F) \times \Sigma_b^k$ to $Q \times (\Sigma_b \cup \{L, R\})^k$, where $L$ and $R$ are the symbols for the right and left moves of the heads respectively. If for each $i$ with $1 \le i \le k$, $x_i y_i$ is the word written on the $i$-th tape (the tape is blank to the right and left of $x_i y_i$) and the machine is looking at the leftmost letter of $y_i$ in state $q$, then the $k$-tuple

$$c = (x_1 q y_1, x_2 q y_2, \ldots, x_k q y_k) \tag{3.1}$$

is a *configuration* of $M$. This describes the current state of the machine $M$. If $\delta(q, a_1, \ldots, a_k) = (q', R, a_2', \ldots, a_k')$, for example, then by one action of $M$, $c$ changes to a new configuration

$$c' = (x_1 a_1 q' y_1', x_2 q' a_2' y_2', \ldots, x_k q' a_k' y_k'),$$

where $y_1 = a_1 y_1'$, $y_2 = a_2 y_2'$, $\ldots$, $y_k = a_k y_k'$. The *size* $|c|$ of the configuration $c$ in (3.1) is defined by

$$|c| = |x_1 y_1 x_2 y_2 \cdots x_k y_k|.$$

For $x \in \Sigma^*$, let $\tau_M(x)$ be the number of steps taken until $M$ halts when it starts from $q_0$ with input $x$ written in the first tape of $M$ (the other tapes are empty). The *time function* $t_M : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ of $M$ is defined by

$$t_M(n) = \max\{\tau_M(x) \mid x \in \Sigma^n\}.$$

For a configuration $c$, let $\tau_M'(c)$ be the number of steps taken until $M$ halts when it starts with $c$. In particular, $\tau_M(x) = \tau_M'(q_0 x, q_0, \ldots, q_0)$ for $x \in \Sigma^*$. Define the *total time function* function $t_M' : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ of $M$ by

$$\begin{aligned} t_M'(n) &= \max\{\tau_M'(c) \mid c : \text{configuration of size } n\} \\ &= \max\{\tau_M'(x_1 q y_1, \ldots, x_k q y_k) \mid q \in Q, x_1 y_1 \cdots x_k y_k \in \Sigma_b^n\}. \end{aligned}$$

Clearly,

$$t_M(n) \le t_M'(n)$$

for any $n \in \mathbb{N}$.

A $Q$-*system* is a finite rewriting system $R$ over an alphabet

$$\Sigma = Q \cup \Sigma_1 \cup \Sigma_2 \cup \{\not\epsilon, \$\} \quad \text{(union of two-by-two disjoint sets)}$$

consisting of rules only of the form

$$\begin{array}{rcll} vqu & \to & v'q'u', & \\ \not\epsilon vqu & \to & \not\epsilon v'q'u', & \\ vqu\$ & \to & v'q'u'\$, & \text{or} \\ \not\epsilon vqu\$ & \to & \not\epsilon v'q'u'\$, & \end{array}$$

where $q, q' \in Q$, $u, u' \in \Sigma_1^*$ and $v, v' \in \Sigma_2^*$.

A word $x \in \Sigma^*$ is *admissible* (resp. *weakly admissible*), if it is of the form $vqu$ with $q \in Q$, $u \in \Sigma_1^*\,\$$ (resp. $u \in \Sigma_1^* \cup \Sigma_1^*\$$) and $v \in \not\epsilon\,\Sigma_2^*$ (resp. $v \in \Sigma_2^* \cup \not\epsilon\,\Sigma_2^*$). Any word $x \in \Sigma^*$ can be uniquely decomposed as

$$x = y_0 x_1 y_1 \cdots x_k y_k, \tag{3.2}$$

where $x_i$ are maximal weakly admissible subwords of $x$ and $y_i$ contains no letter from $Q$. Since any rule of a $Q$-system $R$ can be applied only within a maximal weakly admissible subword, we have

$$\delta_R(x) = \delta_R(x_1) + \cdots + \delta_R(x_k) \tag{3.3}$$

for a word $x$ decomposed as in (3.2).

For a $Q$-system $R$ and for $n \in \mathbb{N}$, define

$$\begin{aligned} ad_R(n) &= \max\{\delta_R(x) \mid x \text{ is admissible and } |x| = n + 3\} \\ &= \max\{\delta_R(\not\epsilon vqu\$) \mid u \in \Sigma_1^*, v \in \Sigma_2^*, q \in Q, |u| + |v| = n\}. \end{aligned}$$

**Lemma 3.1.** *For a $Q$-system $R$, we have*

$$ad_R(n) \le d_R(n + 3)$$

*for any $n \in \mathbb{N}$. If $ad_R$ is super-additive, then*

$$d_R(n + 1) \le ad_R(n)$$

*for any $n \in \mathbb{N}$. If $ad_R$ is equivalent to a nonzero super-additive function, then*

$$d_R(n + 1) = O(ad_R(n)).$$

**Proof.** Since $\delta_R(x) \leq d_R(n+3)$ for any admissible word $x \in \Sigma^{n+3}$, we find $ad_R(n) \leq d_R(n+3)$. Suppose that $ad_R$ is super-additive, and let $x \in \Sigma^*$ be a word of the form (3.2). Let $x_i'$ be the admissible word obtained from $x_i$ by putting the symbols $\not\in$ and \$ if necessary. By (3.3) and the super-additivity of $ad_R$ we have

$$\begin{aligned}\delta_R(x) &= \delta_R(x_1) + \cdots + \delta_R(x_k)\\ &\leq \delta_R(x_1') + \cdots + \delta_R(x_k')\\ &\leq ad_R(|x_1'| - 3) + \cdots + ad_R(|x_k'| - 3)\\ &\leq ad_R(|x_1'| + \cdots + |x_k'| - 3k)\\ &\leq ad_R(|x| - k).\end{aligned}$$

Because $\delta_R(x) = 0$ if $k = 0$, we may only consider the case $k \geq 1$. Thus, we have $d_R(n+1) \leq ad_R(n)$ for any $n \in \mathbb{N}$.

Next suppose that $ad_R$ is equivalent to a super-additive function $f \neq 0$. We may assume that $f$ is positive, that is, $f(n) > 0$ for every $n > 0$, because we can replace it by $f(n) + n = \Theta(f(n))$. Then, there is a constant $C > 0$ such that $ad_R(0) \leq C$ and $ad_R(n) \leq C f(n)$ for every $n > 0$. In the above calculation suppose that $|x_1'| > 3, \ldots, |x_\ell'| > 3$ and $|x_{\ell+1}'| = \cdots = |x_k'| = 3$ for some $\ell \leq k$. Then, we have

$$\begin{aligned}\delta_R(x) &\leq ad_R(|x_1'| - 3) + \cdots + ad_R(|x_\ell'| - 3) + (k - \ell) ad_R(0)\\ &\leq C f(|x_1'| - 3) + \cdots + C f((|x_\ell'| - 3) + C(k - \ell)\\ &\leq C f(|x_1'| + \cdots + |x_\ell'| - 3\ell) + C f(k - \ell)\\ &\leq C f(|x| - \ell).\end{aligned}$$

Here, if $\ell \geq 1$, we have $\delta_R(x) \leq C \cdot f(|x| - 1)$. If $\ell = 0$, we have $\delta_R(x) \leq C\, k \leq C f(|x| - 1)$ for sufficiently long $x$. It follows that $d_R(n+1) = O(f(n)) = O(ad_R(n))$. □

There is a natural way to simulate one-tape Turing machines by string rewriting systems [3]. Let $M = (\Sigma, Q, q_0, F, \delta)$ be a one-tape Turing machine. Here, $\delta$ is a mapping from $(Q \setminus F) \times \Sigma_b$ to $Q \times (\Sigma_b \cup \{L, R\})$. We define a Q-system $R_M$ associated with $M$ as follows. $R_M$ is a rewriting system on the alphabet

$$\Omega = Q \cup \Sigma_b \cup \overline{\Sigma}_b \cup \{\not\in, \$\} \text{ (union of two-by-two disjoint sets)},$$

where $\overline{\Sigma}_b = \{\bar{a} | a \in \Sigma_b\}$ is a copy of $\Sigma_b$, and consists of the rules:

$$\begin{array}{lll} qa \to \bar{a}q' & \text{for} & \delta(q,a) = (q', R),\\ \bar{a}'qa \to q'a'a & \text{for} & \delta(q,a) = (q', L),\\ qa \to q'a' & \text{for} & \delta(q,a) = (q', a'),\\ q\$ \to \bar{b}q'\$ & \text{for} & \delta(q,b) = (q', R),\\ \bar{a}q\$ \to q'a\$ & \text{for} & \delta(q,b) = (q', L),\\ q\$ \to q'a\$ & \text{for} & \delta(q,b) = (q', a),\\ \not\in qa \to \not\in q'ba & \text{for} & \delta(q,a) = (q', L),\\ \not\in q\$ \to \not\in q'b\$ & \text{for} & \delta(q,b) = (q', L)\end{array}$$

for $a, a' \in \Sigma_b$, $q \in Q \setminus F$ and $q' \in Q$.

For a word $x \in \Sigma_b^*$, $\bar{x}$ denotes the word obtained from $x$ by replacing every letter $a$ in $x$ by $\bar{a}$. Since one step of the Turing machine $M$ just corresponds to one rewriting by $R_M$ we have the following result.

**Lemma 3.2.** *It holds that*

$$\delta_{R_M}(\not\in q_0 x\$) = \tau_M(x), \qquad \delta_{R_M}(\not\in \bar{x} q y\$) = \tau_M'(xqy)$$

*for $x, y \in \Sigma_b^*$ and $q \in Q$.*

The following follows from Lemmas 3.1 and 3.2.

**Corollary 3.3.** *We have*

$$d_{R_M}(n+3) \geq ad_{R_M}(n) = t_M'(n) \geq t_M(n)$$

*for $n \in \mathbb{N}$. If $t_M'(n)$ is equivalent to a nonzero super-additive function, then*

$$\Theta(t_M'(n-3)) \leq d_R(n) = O(t_M'(n-1)).$$

If $R$ is finite and terminating, then we can compute $d_R$ by tracing all the derivation sequences (see Section 5), and it is a recursive function. The following corollary asserts that it can be greater than any given recursive function.

**Corollary 3.4.** *For any recursive function $f$, there exists a finite terminating rewriting system $R$ such that*

$$d_R(n) \geq f(n)$$

*for any positive $n \in \mathbb{N}$.*

**Proof.** Let $M$ be a deterministic one-tape Turing Machine computing $f$ ($M$ computes the unary $f(n+3)$ for a given unary $n \in \mathbb{N}$). With input $n$, $M$ runs for at least $f(n+3)$ steps (to print the output $f(n+3)$). Let $R = R_M$ be the rewriting system on $\Omega$ associated with $M$. By Corollary 3.3 we see

$$d_R(n+3) \geq t_M(n) \geq f(n+3)$$

for any $n \in \mathbb{N}$. Adding rules (and letters) to $R$ if necessary, we can make $d_R(n) \geq f(n)$ for $n = 1, 2$ also. □

## 4. Time functions and derivational complexity

As we have seen in the last section, derivational complexity is related to the time functions of Turing machines. In this section we study a more precise relation between them.

Let $M = (\Sigma, Q, q_0, F, \delta)$ be a $k$-tape Turing machine. We assume that the time function $t_M(n)$ of $M$ is at least linear and moreover it satisfies the condition

$$t_M(n) = \Omega\big(n \log t_M(n)\big). \tag{4.1}$$

For a real number $\alpha > 1$ the functions $n^\alpha$ and $\alpha^n$ satisfy this condition.

Define another Turing machine $M_1 = (\Sigma', Q, q_0, F, \delta_1)$ as follows (cf. [2]). $M_1$ has a one-way additional tape called the history tape (so $M_1$ is a $(k+1)$-tape machine). Let $C_M$ is the set of commands of $M$, that is, $C_M$ is the finite subset of $Q \times \Sigma_b^k \times Q \times (\Sigma_b \cup \{L, R\})^k$ given by

$$C_M = \{(q, a, \delta(q, a)) \mid q \in Q \setminus F, a \in \Sigma_b\}.$$

The set $\Sigma' = \Sigma \cup C_M$ is the alphabet for $M_1$. The machine $M_1$ acts as $M$ on the first $k$ tapes, but in each step it writes the command $c \in C_M$ that it executes on the history tape and shifts its head to the right by one.

Next we define another machine $M_2 = (\Sigma', Q_2, q_0, F_2, \delta_2)$ as follows. $M_2$ is also a $(k+1)$-tape machine with a history tape (the last tape). The set $Q_2$ of states contains the sets $Q$ and $F_2$ which consists of two final states $q_s$ and $q_f$. The machine $M_2$ shifts the head of the history tape to the left by one (as far as the head is not at the leftmost position), and then reads a letter $c \in C_M$ on it ($M_2$ halts in $q_f$ if $c$ is not a command). Then, it checks if the configuration consisting of the words on the first $k$ tapes can be obtained from another configuration by an application of the command $c$ as an action of $M$. If this is not the case, $M_2$ halts in $q_f$, otherwise it returns to its previous configuration (this is unique for $c$ and the current configuration) and erases the letter $c$. Repeating this execution, if the head of the history tape reaches the left end, $M_2$ checks if the current state is the initial state $q_0$, the head of the first tape is at the initial position and all the tapes are empty except for the first tape. If so, $M_2$ halts in $q_s$, otherwise it halts in $q_f$.

Define still another Turing machine $M'$ composing $M_1$ and $M_2$ as follows. $M'$ has $2k+2$ tapes and the first $k+1$ tapes are for the machine $M_1$ and the second $k+1$ tapes are for $M_2$. $M'$ acts as $M_1$ and as $M_2$ alternatively. After $M'$ acts one step as $M_1$ on the first $k+1$ tapes, then $M'$ acts as $M_2$ on the last $k+1$ tapes taking a sequence of steps as described above (reads a command and reverses the configuration by one step). If $M_1$ halts or $M_2$ halts in $q_f$, then $M'$ halts. If $M_2$ halts in $q_s$, $M'$ copies the new data on the first $k$ tapes to the second $k$ tapes (including the head positions), copies the data from the first history tape to the position to the head onto the second history tape, and sets the head of the last tape just after the copied data. Then again $M'$ continues to take a step of $M_1$ (and $M_2$ alternatively).

Now we claim that the time functions of $M$ and $M'$ are equivalent, and the time function and the total time function of $M'$ are equivalent.

If $M'$ starts with a configuration $(q_0x, q_0, \dots q_0)$ with $x \in \Sigma^*$ in the first tape and the other tapes empty, then $M'$ copies the data $x$ in the first tape to the $(k+2)$-nd tape, and then $M'$ acts as $M$ starting with the initial configuration with input $x$ doing additional works, recording commands on the first history tape etc. Hence,

$$t_M(n) \leq t_{M'}(n) \leq t'_{M'}(n). \tag{4.2}$$

Now suppose that $M'$ starts with an arbitrary configuration

$$c = (x_1qy_1, \dots, x_kqy_k, xqy, u_1pv_1, \dots, u_kpv_k, upv),$$

where the pair $(q, p)$ is a state of $M'$. If $M_2$ halts in $q_f$ starting with the configuration $(u_1pv_1, \dots, u_kpv_k, upv)$, then $M'$ halts during reading $u$ or just after reading all letters in $u$, and we have

$$\tau'_{M'}(c) = O(|u|) = O(|c|) = O(t_M(|c|)).$$

On the other hand if $M_2$ halts in $q_s$, then the current configuration of $M'$ is of the form

$$(x'_1q'y'_1, \dots, x'_kq'y'_k, x'q'y', q_0z', q_0, \dots, q_0).$$

Because $M_1$ and $M_2$ act alternatively ($M_1$ acts first), $x' = xw$ for some $w \in C_M^*$ with $|w| = |u| + 1$. Hence $|x'| = |x| + |u| + 1$ and

$$\ell' = |x'_1y'_1 \cdots x'_ky'_k| \leq \ell + k \cdot (|u| + 1)$$

hold, where $\ell = |x_1y_1 \cdots x_ky_k|$. Then, $M'$ writes $x'$ in the second history tape and copies the data of the total size $\ell'$ from the first $k$ tapes to the second $k$ tapes. Then, $M'$ gets into a new loop to take steps of $M_1$ and $M_2$ alternatively. If this new cycle is successfully completed, the configuration $(x'_1q'y'_1, \ldots, x'_kq'y'_k)$ comes from a configuration $(q_0z, q_0, \ldots, q_0)$ with some $z \in \Sigma'^*$ by the execution of the commands in $x' = xw$. In particular the configuration $(x_1qy_1, \ldots, x_kqy_k)$ comes from the configuration $(q_0z, q_0, \ldots, q_0)$ by the execution of the commands in $x$. Hence we see $|z| \le |x_1y_1| + |x| \le |c|$ and $\ell \le |z| + k|x|$. Since $x'' = x'w'$ for $w' \in C_M^*$ with $|w'| = |x'| + 1$, we have $|x''| = 2|x'| + 1$.

Suppose that $M'$ halts after repeating $m$ ($\ge 2$) loops. In every $i$-th loop, $|x^{(i-1)}| + 1$ commands are executed forwards on the first $k$ tapes and $|x^{(i-1)}|$ commands are executed backwards on the second $k$-tapes, and the data $x^{(i)}$ on the first history tape and the other $\ell^{(i)}$ letters are copied. We have $|x^{(i+1)}| = 2|x^{(i)}| + 1$ and $\ell^{(i+1)} \le \ell^{(i)} + k \cdot (|(x^{(i)}| + 1)$ for $i \ge 1$. It follows that

$$|x^{(i+1)}| = 2^i(|x'| + 1) - 1 = 2^i(|x| + |u| + 2) - 1 \tag{4.3}$$

and

$$\ell^{(i+1)} \le \ell^{(i)} + 2^{i-1}k \cdot (|x| + |u| + 2) \le \ell' + (2^i - 1)k \cdot (|x| + |u| + 2) < |z| + 2^ik \cdot (|x| + |u| + 2) \tag{4.4}$$

for $i \ge 0$. Because $M$ takes at least $|x^{(m)}|$ steps starting with input $z$, we see

$$t_M(|z|) \ge |x^{(m)}| = 2^{m-1}(|x| + |u| + 2) - 1 \tag{4.5}$$

for $m \ge 2$. Hence $\log_2 t_M(|z|) > m - 1$. Since $M'$ halts before completing the $(m+1)$-th loop, using (4.1) and (4.3)–(4.5) we have

$$\begin{aligned}
\tau'_{M'}(c) &= O\big(|u| + |x'| + |x''| + \cdots + |x^{(m)}| + \ell' + \ell'' + \cdots + \ell^{(m)}\big) \\
&= O\big(m \cdot |z| + 2^m(k+1) \cdot (|x| + |u| + 2)\big) \\
&= O\big(|z| + |z| \cdot \log_2 t_M(|z|) + 2(k+1) \cdot t_M(|z|)\big) \\
&= O(t_M(|z|)) = O(t_M(|c|)).
\end{aligned}$$

Hence, we have

$$t'_{M'}(n) = O(t_M(n)). \tag{4.6}$$

By (4.2) and (4.6) we obtain

$$t'_{M'}(n) = \Theta(t_{M'}(n)) = \Theta(t_M(n)).$$

It is well-known that any multi-tape Turing machine can be simulated by a one-tape Turing machine (see [8] for example). Here we need additional functions for correct synchronization. Let $k \ge 2$ and $M = (\Sigma, Q, q_0, F, \delta)$ be a $k$-tape Turing machine. Define a one-tape Turing machine $M' = (\Sigma', Q', q'_0, F', \delta')$ as follows. Let $\Sigma' = \Sigma \cup \{\#, \bar{b}\}$ and $Q'$ contain $Q$. The tape of $M'$ is divided into $2k$ tracks. In the $(2i-1)$-th track ($1 \le i \le k$), the data in the $i$-th tape of $M$ are written and in the $2i$-th track the head marker # is written. The process of the machine is divided into two phases. In the first phase $M'$ scans the tape from left to right. If $M'$ reads the letter # in the $2i$-th track, it records the letter (say $a$) at the position of # in the $2i-1$-th track. If $a = \#$, $M'$ halts, and if $a = b$, replace $b$ by $\bar{b}$. If $M'$ reads $b$, $M'$ always replaces it by $\bar{b}$. If $M'$ reads consecutive $2k$ blanks, then it check if all the letters corresponding to the positions of the mark # are read. If so (otherwise it halts), $M'$ enters the second phase. In the second phase the machine scans the tape from right to left. If $M'$ reads # in the $2i$-th track, it acts on the $2i-1$-th track as $M$ acts on the $i$-th tape (if it reads $\bar{b}$ on the tape, it acts as if it reads $b$). For example, if $M$ prints a letter $a$ in the $i$-th tape, $M'$ prints $a$ in the $(2i-1)$-th track at the position the mark # in the $2i$-th track indicates. If the machine finds any inconsistency, it halts. If the head reaches the leftmost position, the machine again enters the first phase.

We claim that the total time function of $M'$ is equivalent to the square of the total time function of $M$.

For an arbitrary configuration $c = (x_1qy_1, \ldots, x_kqy_k)$ of $M$, let $c' = qy$ be the configuration of $M'$ corresponding to $c$ with the state $q$ in the initial position and the word $y$ in $\Sigma'^*$. Start $M'$ with the configuration $c'$ in the first phase, then $M'$ starts to simulate $M$, and for one step of $M$, $M'$ scans the tape in both ways and does the work for $M$ and moreover reads $2k$ $b$'s and replaces them by $2k$ $\bar{b}$'s in the right end of the tape. Thus for one step of $M$, the number of steps $M'$ executes is between $2(|c| + k)$ and $C(|c| + k)$ for some constant $C > 2$, and the increase of the size of the configuration is between $2k$ and $4k$ (including the increase in the left end of the tape). So, till $M$ halts, the number of steps $M'$ executes is between $C_1(|c| + 2k + |c| + 4k + \cdots + |c| + 2k \cdot \tau'_M(c))$ and $C_2(|c| + 4k + |c| + 8k + \cdots + |c| + 4k \cdot \tau'_M(c))$ for some constants $C_1, C_2 > 0$. So it executes

$$\Theta(|c| \cdot \tau'_M(c)) + \Theta(1 + 2 + \cdots + \tau'_M(c)) = \Theta(\tau'_M(c)^2)$$

steps. Now, let $c' = xq'y$ ($q' \in Q', x, y \in \Sigma'^*$) be any configuration of $M'$. If $M'$ halts before reaching a configuration corresponding to a configuration of $M$, then $\tau'_{M'}(c') \le O(|c'|)$. On the other hand if it gets to a configuration corresponding to a configuration $c$ of $M$, $M'$ starts to simulate $M$ with the configuration $c$ as above. Our discussion shows that $\Theta(t'_{M'}) = \Theta((t'_M)^2)$.

Suppose that $f$ is the time function of a $k$-tape Turing machine $M$ such that $f(n) = \Omega(n \log f(n))$ and $f^2$ is equivalent to a nonzero super-additive function $g$. Let $M'$ be the $(2k+2)$-tape machine constructed above from $M$ in the first paragraph. We have

$$f(n) = \Theta(t_{M'}(n)) = \Theta(t'_{M'}(n)).$$

Let $M''$ be the one-tape Turing machine constructed from $M'$ in the second paragraph above. We have

$$t'_{M''}(n) = \Theta(t'_{M'}(n)^2) = \Theta(f(n)^2) = \Theta(g(n)).$$

Let $R$ be the $Q$-system associated with $M''$, then by Corollary 3.3, we see

$$\Theta(f(n-3)^2) = \Theta(t'_{M''}(n-3)) \le d_R(n) = O(t'_{M''}(n-1)) = O(f(n-1)^2).$$

Thus, we have

**Theorem 4.1.** *Let $f(n)$ be a time function of a Turing machine such that $f(n) = \Omega(n \log f(n))$ and $f(n)^2$ is equivalent to a nonzero super-additive function. Then there exists a finite rewriting system $R$ such that*

$$\Theta(f(n-3)^2) \le d_R(n) = O(f(n-1)^2).$$

We say that a function $f : \mathbb{N} \to \mathbb{N}$ is computable in time $O(g(n))$, if there exists a (deterministic) algorithm computing $f(n)$ within time $O(g(n))$, more precisely, if there exists a multi-tape Turing machine $M$ which computes binary $f(n)$ for given binary $n$ with time function $t_M(n) = O(g(n))$.

**Lemma 4.2.** *If $f : \mathbb{N} \to \mathbb{N}$ is a function such that $f(n) = \Omega(n^2)$ and the binary $f(n)$ is computable in time $O(\sqrt{f(n)})$ for binary $n \in \mathbb{N}$, then $\lfloor \sqrt{f(n)} \rfloor$ is equivalent to the time function of a Turing machine.*

**Proof.** Given unary $n \in \mathbb{N}$, compute binary $n$ in time $O(n) \le O(\sqrt{f(n)})$. Then, compute binary $f(n)$ in time $O(\sqrt{f(n)})$. Next compute binary $\lfloor \sqrt{f(n)} \rfloor$ in time $O((\log_2 f(n))^3) \le O(\sqrt{f(n)})$. Finally, transform the binary $\lfloor \sqrt{f(n)} \rfloor$ to unary in time $O(\sqrt{f(n)})$. The total time for computing unary $\lfloor \sqrt{f(n)} \rfloor$ is $O(\sqrt{f(n)})$. This means that there is a (multi-tape) Turing machine with time function equivalent to $\lfloor \sqrt{f(n)} \rfloor$. □

Combining this lemma with Theorem 4.1 we have

**Theorem 4.3.** *Suppose that a function $f(n) = \Omega(n^2 \log^2 f(n))$ is computable in time $O(\sqrt{f(n)})$ in binary and equivalent to a nonzero super-additive function. Then, there exists a finite rewriting system $R$ such that*

$$\Theta(f(n-3)) \le d_R(n) = O(f(n-1)).$$

## 5. Computing the derivational complexity

Let $R$ be a finite rewriting system on $\Sigma$. Consider a derivation sequence of length 2:

$$x = x'ux'' \to_R x'vx'' = y = y'u'y'' \to_R y'v'y'' = z,$$

where $u \to v, u' \to v' \in R$. This sequence is *left canonical*, if

$$|x'| < |y'u'|.$$

A sequence is *left canonical*, if every subsequence of length 2 of it is left canonical. In particular, a sequence of length $\le 1$ is left canonical. Let

$$p : x_0 \to_R x_1 \to_R x_1 \to_R \cdots \to_R x_n \tag{5.1}$$

be a derivation sequence with respect to $R$. If $x_{i-1} \to_R x_i \to_R x_{i+1}$ is not left canonical, that is,

$$x_{i-1} = x'ux''u'x^\dagger, \qquad x_i = x'ux''v'x^\dagger, \qquad x_{i+1} = x'vx''v'x^\dagger,$$

with $x', x'', x^\dagger \in \Sigma^*$ and $u \to v, u' \to v' \in R$, then we replace this part by the sequence $x_{i-1} \to_R x'_i \to_R x_{i+1}$, where $x' = x'vx''u'x^\dagger$. Repeating this modification as long as the sequence is not left canonical, we finally get a left canonical sequence of the same length from $x_0$ to $x_n$. Thus we have

**Lemma 5.1** (*c.f. [5]*)**.** *For a derivation sequence of length $n$ from $x \in \Sigma^*$ to $y \in \Sigma^*$, there is a left canonical sequence from $x$ to $y$ of the same length $n$.*

For a sequence (5.1) we define a number $L(p)$ by induction on $n$ as follows. When $n = 1$ and $p : x_0 = x'_0ux''_0 \to_r x'_0vx''_0 = x_1$ with $r = (u \to v) \in R$, define

$$L(p) = |x'_0u| = |x_0| - |x''_0|. \tag{5.2}$$

Suppose that $n \geq 2$ and

$$x_{n-2} = x'_{n-2}u'x''_{n-2} \to_{r'} x'_{n-2}v'x''_{n-2} = x_{n-1} = x'_{n-1}ux''_{n-1} \to_r x'_{n-1}vx''_{n-1} = x_n \tag{5.3}$$

with $r = (u \to v), r' = (u' \to v') \in R$. Then, define

$$L(p) = L(p') + |x'_{n-1}| - |x'_{n-2}| + |u| + K - 1,$$

where $p'$ is the subsequence

$$x_0 \to_R x_1 \to_R \cdots \to_R x_{n-1} \tag{5.4}$$

of $p$ and

$$K = \max\{|u|, |v| \mid u \to v \in R\}.$$

**Lemma 5.2.** *For any derivation sequence $p$ of length $n$ ($\geq 1$) starting with $x \in \Sigma^*$ we have*

$$L(p) \leq (2K-1)(n-1) + |x|.$$

**Proof.** Let $p$ be a sequence (5.1) with the tail (5.3). We shall prove the stronger inequality

$$L(p) \leq (2K-1)(n-1) + |x_0| - |x''_{n-1}| \tag{5.5}$$

by induction on $n$. First, (5.2) implies that the induction start is cleared. Let $n \geq 2$ and $p'$ be the subsequence (5.4). By induction hypothesis

$$L(p') \leq (2K-1)(n-2) + |x_0| - |x''_{n-2}|.$$

Since $x'_{n-2}v'x''_{n-2} = x'_{n-1}ux''_{n-1}$, we have

$$\begin{aligned} L(p) &\leq (2K-1)(n-2) + |x_0| - |x''_{n-2}| + |x'_{n-1}| - |x'_{n-2}| + |u| + K - 1 \\ &= (2K-1)(n-2) + |x_0| - |x''_{n-1}| + |v'| + K - 1 \\ &\leq (2K-1)(n-1) + |x_0| - |x''_{n-1}|, \end{aligned}$$

completing the induction step to prove (5.5). □

To find a derivation sequence starting with a word $x$, we scan the word from the left end. If we find a subword $u$ with $u \to v \in R$ ($x = x'ux''$), we replace it by $v$. Suppose that we scan the word $y = x'vx''$ from the $i$-th letter of $x'$ and find a subword $u'$ with $u' \to v' \in R$ ($y = y'u'y''$) and replace it by $v'$. Then, we get a sequence $x \to_R y \to_R y'v'y''$ of length 2 by tracing $|x'| + |u| + |y'| - i + 1 + |u'|$ letters.

**Lemma 5.3.** *Any left canonical derivation sequence $p$ can be found by tracing at most $L(p)$ letters in the words appearing in $p$.*

**Proof.** Suppose that the subpath $p'$ in (5.4) is found by tracing $L(p')$ letters. Let $i = \max\{|x'_{n-2}| - K + 2, 1\}$. To get a left canonical sequence $p$ extending $p'$, we restart the tracing from the $i$-th letter in $x'_{n-2}$. Then reading at most $|x'_{n-1}| - |x'_{n-2}| + |u| + K - 1$ letters in $x_{n-1}$, we find a left-hand side $u$ of a rule of $R$. So, we traced at most $L(p)$ letters to find $p$. □

**Theorem 5.4.** *Let $R$ be a finite rewriting system on $\Sigma$ with derivational complexity $f$. Then, $f(n)$ can be computed deterministically in time $C^{f(n)}$ for some constant $C > 1$.*

**Proof.** With a left canonical sequence $p$ in (5.1) we associate a list $\mathcal{L}_1(p)$ of words and a list $\mathcal{L}_2(p)$ of rules of $R$ as follows. When $n = 1$ and $p : x_0 = x'_0u'x''_0 \to_{r'} x'_0v'x''_0$ with $r' = u' \to v' \in R$, then $\mathcal{L}_1(p) = (x'_0)$, and $\mathcal{L}_2(p) = (r')$. Suppose that $n \geq 2$ and $\mathcal{L}_1(p')$ and $\mathcal{L}_2(p')$ are defined for the subsequence $p'$ (5.4), then for $p$ with a tail (5.3), $\mathcal{L}_1(p)$ and $\mathcal{L}_2(p)$ are defined by appending the word $y$ and the rule $r$ to $\mathcal{L}_1(p')$ and $\mathcal{L}_2(p')$ respectively, where $y$ is the suffix of $x'_{n-1}$ of length $\min\{|x'_{n-1}|, |x'_{n-1}| - |x'_{n-2}| + K - 1\}$ and $r = u \to v$. Clearly, the lengths of $\mathcal{L}_1(p)$ and $\mathcal{L}_2(p)$ is equal to the length $n$ of $p$, and $L_2(p)$ is just the list of the rules appearing in $p$. Moreover, by the definitions of $L(p)$ and $\mathcal{L}_1(p)$ and by the discussion in the proof of Lemma 5.3, we see that

(i) the initial word $x_0$ and the lists $\mathcal{L}_1(p)$ and $\mathcal{L}_2(p)$ determine the sequence $p$, and

(ii) the sum of the length of words in $\mathcal{L}_1(p)$ does not exceed $L(p)$.

For $N, L > 0$ let $S(N, L)$ denote the set of all lists $\ell$ of words over $\Sigma$ such that the length of $\ell$ is bounded by $N$ and the sum of lengths of words in $\ell$ is bounded by $L$. Since such lists are coded with words over $\Sigma \cup \{,\}$ (disjoint union) of length $\leq N+L-1$, we see $|S(N, L)| < (|\Sigma| + 1)^{N+L}$. For a left canonical sequence $p$ of length $\leq n$, $\mathcal{L}_1(p)$ is in $S(n, L(p)) \subset S(n, (2K-1)n + m)$ by Lemma 5.2 and (ii) above, where $m = |x_0|$. The mapping which sends a left canonical sequence to the pair $(\mathcal{L}_1(p), \mathcal{L}_2(p))$ of the lists is injective by (i). The length of a left canonical sequence starting $p$ with a word $x_0$ of length $m$ is bounded by $f(m)$. Hence, $\mathcal{L}_1(p)$ is in $S(f(m), (2K-1)f(m) + m)$ and $\mathcal{L}_2(p)$ is a list of elements of $R$ of length $\leq f(m)$. Thus, the total number of such left canonical sequence are bounded by

$$(|\Sigma| + 1)^{2Kf(m)+m} \cdot |R|^{f(m)+1} < C_1^{f(m)}$$

with some constant $C_1 > 1$. There are $|\Sigma|^m$ words over $\Sigma$ of length $m$ and we can find one left canonical sequence starting with a word of length $m$ in time $O(f(m))$ by Lemmas 5.2 and 5.3. So we can compute all the lengths of left canonical sequences starting words of length $m$ and the maximum of them in time

$$O(f(m) \cdot |\Sigma|^m \cdot C_1^{f(m)}) \leq C^{f(m)}$$

for some constant $C > 1$. □

**Corollary 5.5.** *Let $f(n)$ be a recursive function whose computational complexity exceeds $C^{f(n)}$ for any $C > 0$, then there exists no finite rewriting system with derivational complexity equivalent to $f(n)$.*

## 6. Complexities of the forms $n^\alpha$ and $\alpha^n$

In this section we shall show that there are finite rewriting systems with derivational complexities equivalent to $n^\alpha$ (and $\alpha^n$), if the computational complexity of the real number $\alpha$ is relatively low, but there are no such systems if the complexity of $\alpha$ is high. The author has been inspired by the discussions in [12], where a similar result for the function $n^\alpha$ on the Dehn functions of finitely presented groups is given.

A real number $\alpha > 0$ is *computable in time $f(n)$*, if a binary rational approximation $a/b$ ($a, b \in \mathbb{N}$) of $\alpha$ such that $b = O(2^n)$ and

$$\left|\alpha - \frac{a}{b}\right| < \frac{1}{2^n} \tag{6.1}$$

can be computed in time $O(f(n))$ (refer to [13] for computable real numbers). We denote this rational $a/b$ by $\alpha[n]$ (A rational $a/b$ satisfying (6.1) and $b = O(2^n)$ may not be unique, but we fix a deterministic algorithm that outputs a rational satisfying the conditions in time $O(f(n))$ for an input $n$. The rational $\alpha[n]$ is this output computed by the algorithm.)

If the binary expression of $\alpha$ up to $n$ digits (after radix point) can be computed in time $O(f(n))$, then $\alpha$ is computable in time $f(n)$, because $\lfloor 2^n\alpha \rfloor / 2^n$ is a rational approximation of $\alpha$ with error $< 1/2^n$.

**Lemma 6.1.** *Let $\alpha > 0$ be a real number computable in time $f(n)$. Then for an integer $\nu$, the function*

$$g_{\alpha,\nu}(n) = 2^{\lfloor \alpha[\lceil \log_2 n \rceil - \nu] \cdot n \rfloor} \ (n \geq 2^\nu)$$

*is equivalent to $2^{\alpha n}$ and can be computed in time $O(f(\lceil \log_2 n \rceil - \nu) + n)$.*

**Proof.** Let $m = \lceil \log_2 n \rceil$. Since $n \leq 2^m$, by (6.1) we have

$$|\alpha \cdot n - \alpha[m - \nu] \cdot n| < \frac{n}{2^{m-\nu}} \leq 2^\nu.$$

Hence,

$$\lfloor \alpha[m - \nu] \cdot n \rfloor - 2^\nu < \alpha n < \lfloor \alpha[m - \nu] \cdot n \rfloor + 1 + 2^\nu,$$

and so,

$$2^{-2^\nu} g_{\alpha,\nu}(n) < 2^{\alpha n} < 2^{1+2^\nu} g_{\alpha,\nu}(n),$$

implying that $2^{\alpha n}$ and $g_{\alpha,\nu}(n)$ are equivalent. The binary expansion of $m = \lceil \log_2 n \rceil$ is computable in time $O(m)$, and $\alpha[m - \nu]$ is computable in time $O(f(m - \nu))$ by assumption. The product $\lfloor \alpha[m - \nu] \cdot n \rfloor$ is computable in time $O(m^2)$ because the (binary) sizes of $\alpha[m - \nu]$ and $n$ are $O(m)$. Finally, printing the output in time $O(n)$, $g_{\alpha,\nu}(n)$ can be computed in time $O(m) + O(f(m - \nu)) + O(m^2) + O(n) = O(f(m - \nu) + n)$. □

**Theorem 6.2.** *Let $\alpha > 2$ be a real number computable in time $C^{2^n}$ for some constant $C > 1$. Then, there is a finite rewriting system $R$ with derivational complexity equivalent to $n^\alpha$.*

**Proof.** Let $m = \lfloor \log_2 n \rfloor$ and $\nu > 0$. The function $n^\alpha = 2^{\alpha \log_2 n}$ is equivalent to $2^{\alpha m}$. By Lemma 6.1, $2^{\alpha m}$ is equivalent to

$$h(n) = g_{\alpha,\nu}(m) = 2^{\lfloor \alpha[\lceil \log_2 m \rceil - \nu] \cdot m \rfloor}$$

which is computable in time

$$O(C^{2^{\lceil \log_2 m \rceil - \nu}} + m) = O(C^{2^{1-\nu+\log_2 \log_2 n}} + \log_2 n) = O(n^{(\log_2 C)/2^{\nu-1}}).$$

Here, choose $\nu$ large enough to satisfy $(\log_2 C)/2^{\nu-1} \leq \alpha/2$, then we can compute $h(n)$ in time $O(n^{\alpha/2}) \leq O(\sqrt{h(n)})$. Note that the function $n^\alpha$ is super-additive and $n^\alpha = \Omega(n^2 \log^2 n^\alpha)$. Hence, by Theorem 4.3, there is a finite rewriting system $R$ such that

$$\Theta(n^\alpha) = \Theta((n-3)^\alpha) = \Theta(h(n-3)) \leq d_R(n) = O(h(n-1)) = O(n^\alpha). \quad \square$$

Next, we consider the exponential function $\alpha^n$. Though it is not super-additive, it is equivalent to the super-additive function $\alpha^n - 1$ if $\alpha > 1$. In fact, we have

$$\alpha^n - 1 < \alpha^n < 2(\alpha^n - 1)$$

for $n > 1/\log_2 \alpha$, and

$$\alpha^{m+n} - 1 - (\alpha^m - 1) - (\alpha^n - 1) = (\alpha^m - 1)(\alpha^n - 1) \geq 0$$

for $m, n \geq 0$.

Naturally, the computational complexities of $\alpha$ and $\log_2 \alpha$ are closely related.

**Lemma 6.3.** *Let $\alpha$ ($> 1$) be a real number computable in time $f(n)$. Then, $\log_2 \alpha$ is computable in time $f(n+4) + 4^n n^2$, and $2^\alpha$ is computable in time $f(n + \lceil \alpha \rceil + 3) + 8^n n^2$.*

**Proof.** Since $\alpha$ is computable in time $f(n)$, we can compute $\alpha[n+\nu]$ in time $O(f(n+\nu))$ for $\nu \in \mathbb{N}$. Let

$$a_n = \lceil \alpha[n+\nu] \cdot 2^{n+\nu} \rceil.$$

Since

$$\alpha[n+\nu] - \frac{1}{2^{n+\nu}} < \alpha < \alpha[n+\nu] + \frac{1}{2^{n+\nu}}$$

by (6.1), we have

$$a_n - 2 < 2^{n+\nu}\alpha < a_n + 1. \tag{6.2}$$

Letting $\nu = 4$, we have

$$\log_2 a_n + \log_2 \left(1 - \frac{2}{a_n}\right) = \log_2 (a_n - 2) < \log_2 \alpha + n + 4 < \log_2 (a_n + 1) = \log_2 a_n + \log_2 \left(1 + \frac{1}{a_n}\right).$$

Note that $\log_2(1+\epsilon) > 2\epsilon$ for $-1/2 < \epsilon < 0$, and $\log_2(1+\epsilon) < 2\epsilon$ for $\epsilon > 0$. Because $a_n > 2^{n+4} - 1 > 2^{n+3}$ by (6.2), we see that both $-\log_2 (1 - 2/a_n)$ and $\log_2 (1 + 1/a_n)$ are less than $1/2^{n+1}$. Thus,

$$|\log_2 \alpha - (\log_2 a_n - n - 4)| < \frac{1}{2^{n+1}}. \tag{6.3}$$

We can compute $a_n$ from $\alpha[n+4]$ in time $O(n^2)$. Further, we can compute $a_n^{2^{n+1}}$ from $a_n$ in time $O(4^n n^2)$. In fact, for $0 \leq i \leq n$ if $a_n^{2^i}$ is already computed, then we can compute $a_n^{2^{i+1}} = (a_n^{2^i})^2$ in time $O((2^i \log_2 a_n)^2) = O(4^i n^2)$. Thus, in total $a_n^{2^{n+1}}$ can be computed in time

$$O((1 + 4 + \cdots + 4^n) n^2) = O(4^n n^2).$$

Hence we can compute $p_n = \max\{\, i \in \mathbb{N} \mid 2^i \leq a_n^{2^{n+1}} \}$ in time $O(4^n n^2)$. By the definition of $p_n$, we see $2^{p_n} \leq a_n^{2^{n+1}} < a_n^{p_n+1}$ or $p_n \leq 2^{n+1} \log_2 a_n < p_n + 1$. Hence we have

$$\left|\log_2 a_n - \frac{p_n}{2^{n+1}}\right| < \frac{1}{2^{n+1}}. \tag{6.4}$$

Combining (6.3) and (6.4) we have

$$\left|\log_2 \alpha - \left(\frac{p_n}{2^{n+1}} - n - 4\right)\right| < \frac{1}{2^n},$$

and we get the rational approximation $\frac{p_n - 2^{n+1}(n+4)}{2^{n+1}}$ of $\log_2 \alpha$ within the desired time.

Next, we prove the second assertion. With the same $a_n$ as above let

$$b_n = 2^{a_n/2^{n+\nu}}.$$

From (6.2) we have

$$b_n \cdot 2^{-2/2^{n+\nu}} < 2^\alpha < b_n \cdot 2^{1/2^{n+\nu}}.$$

Since $2^\epsilon > 1 + \epsilon$ for $\epsilon < 0$ and $2^\epsilon < 1 + \epsilon$ for $0 < \epsilon < 1$, we see

$$b_n \cdot (1 - 1/2^{n+\nu-1}) < 2^\alpha < b_n \cdot (1 + 1/2^{n+\nu}). \tag{6.5}$$

Here, let $\nu = \lceil \alpha \rceil + 3$. Because $a_n/2^{n+\nu} < \alpha + 1$ by (6.2), we have

$$\log_2 (b_n/2^{n+\nu-1}) = a_n/2^{n+\nu} - n - \nu + 1 < \alpha + 1 - n - \nu + 1 \leq -n - 1.$$

Thus, it follows from (6.5) that

$$|2^\alpha - b_n| < \frac{1}{2^{n+1}}. \tag{6.6}$$

For $n \in \mathbb{N}$, $a_n$ is computed in time $O(f((n+\nu)+n^2)$, and $i^{2^{n+\nu}}$ is computed in time $O(4^n n^2)$ for an integer $i = O(2^n)$, as discussed in the first part of the proof. Let

$$p_n = \max\{i \in \mathbb{N} \mid i^{2^{n+\nu}} \le 2^{a_n+(n+1)2^{n+\nu}}\}.$$

This $p_n$ can be computed in time $O(4^n n^2 \cdot 2^n) = O(8^n n^2)$ by computing $i^{2^{n+\nu}}$ for $i = 2^{n+1}, 2^{n+1}+1, \ldots, 2^{n+1+\lceil\alpha\rceil}$. We have $p_n^{2^{n+\nu}} \le 2^{a_n+(n+1)2^{n+\nu}} < (p_n+1)^{2^{n+\nu}}$, or

$$\frac{p_n}{2^{n+1}} \le b_n < \frac{p_n+1}{2^{n+1}}. \tag{6.7}$$

By (6.6) and (6.7) we have

$$\left|2^\alpha - \frac{p_n}{2^{n+1}}\right| < \frac{1}{2^n}.$$

Thus, $2^\alpha$ is computable in time $f(n+\lceil\alpha\rceil+3)+8^n n^2$. □

If we use a faster algorithm to compute the product of two integers, for example, the Schönhage–Strassen algorithm (see [1]), we can improve Lemma 6.3, but this is enough for our purpose in this paper.

**Corollary 6.4.** *If a real number $\alpha$ ($>1$) is computable in time $f(n) = \Omega(C^n)$ for some $C > 1$, then $\log_2 \alpha$ and $2^\alpha$ are computable in time $f(n+k)$ for some $k \in \mathbb{N}$.*

**Theorem 6.5.** *If a real number $\alpha > 1$ is computable in time $C^{2^n}$ for some constant $C > 1$, then there is a finite rewriting system $R$ with derivational complexity equivalent to $\alpha^n$.*

**Proof.** Since $\alpha$ is computable in time $C^{2^n}$, $\beta = \log_2 \alpha$ is computable in time $C^{2^{n+k}} = C_1^{2^n}$ with $k \in \mathbb{N}$ and $C_1 = C^{2^k}$ by Corollary 6.4. Hence, $\alpha^n = 2^{\beta n}$ is equivalent to $h(n) = g_{\beta,\nu}(n) = 2^{\lfloor\beta[\lceil\log_2 n\rceil-\nu]\cdot n\rfloor}$ which is computable in time

$$O(C_1^{2^{\lceil\log_2 n\rceil-\nu}} + n) \le O(C_1^{2^{1-\nu+\log_2 n}} + n) = O(C_1^{n/2^{\nu-1}})$$

by Lemma 6.1. Here, choose $\nu$ so that $C_1 \le \alpha^{2^{\nu-2}}$, then $h(n)$ is computable in time $O(\alpha^{n/2}) \le O(\sqrt{h(n)})$. Moreover, $h(n)$ is equivalent to the super-additive function $\alpha^n - 1$ as we remarked before Lemma 6.3. Now by Theorem 4.3 there is a finite rewriting system $R$ such that $d_R(n) = O(\alpha^n)$ as in the proof of Theorem 6.2. □

By our results we see that the functions $n^\alpha (\alpha \ge 2)$ and $\alpha^n (\alpha > 1)$ for a rational (or more generally an algebraic) number $\alpha$ are equivalent to the derivational complexities of finite rewriting systems. For a transcendental number $\alpha$ with low complexity such as $\pi$ and $e$, they are also equivalent to the derivational complexities.

Next we discuss the other direction.

**Theorem 6.6.** *Let $\alpha > 1$ be a real number. If there is a finite rewriting system with derivational complexity equivalent to $n^\alpha$ or $\alpha^n$, then $\alpha$ is computable in time $C^{C^{2^n}}$ for some constant $C > 1$.*

**Proof.** Suppose that a rewriting system $R$ has derivational complexity equivalent to $n^\alpha$, that is, there are positive constants $A_1 < A_2$ such that

$$A_1 n^\alpha \le d_R(n) \le A_2 n^\alpha$$

for sufficiently large $n \in \mathbb{N}$. Let $K$ be an integer greater than $1/A_1$ and $\sqrt{A_2}$. Then, we have

$$-1 < \log_K A_1 \le \log_K d_R(K^{2^{n+1}}) - \alpha 2^{n+1} \le \log_K A_2 < 2. \tag{6.8}$$

By Theorem 5.4, $d_R(K^{2^{n+1}})$ can be computed in time $B^{d_R(K^{2^{n+1}})} \le B^{A_2(K^{2^{n+1}})^\alpha}$ for some $B > 1$. So, the binary $p(n) = \lfloor\log_K d_R(K^{2^{n+1}})\rfloor$ is computed also in time $O\big((B^{A_2})^{(K^{2\alpha})^{2^n}}\big)$. From (6.8) we have

$$-2 < p(n) - \alpha 2^{n+1} < 2,$$

or

$$\left|\alpha - \frac{p(n)}{2^{n+1}}\right| < \frac{1}{2^n}.$$

This implies that $\alpha$ is computable in time $C^{C^{2^n}}$ for some $C > 1$.

Next, suppose that a rewriting system $R$ has derivational complexity equivalent to $\alpha^n$, that is, there are positive constants $A_1 < A_2$ such that

$$A_1\alpha^n \le d_R(n) \le A_2\alpha^n$$

for sufficiently large $n \in \mathbb{N}$. Let $K$ be a nonnegative integer greater than $\log_2(1 - \log_2 A_1)$ and $\log_2 \log_2 A_2$. Then, we have

$$1 - 2^K < \log_2 A_1 \le \log_2 d_R(2^{n+K}) - 2^{n+K}\log_2\alpha \le \log_2 A_2 < 2^K. \tag{6.9}$$

By Theorem 5.4, $d_R(2^{n+K})$ can be computed in time $B^{d_R(2^{n+K})} \le B^{A_2\alpha^{2^{n+K}}}$ for some $B > 1$. So, the binary $p(n) = \lfloor \log_2 d_R(2^{n+K}) \rfloor$ is computed in time $O\big((B^{A_2})^{(\alpha^{2^K})^{2^n}}\big)$. From (6.9) we have

$$-2^K < p(n) - 2^{n+K}\log_2\alpha < 2^K$$

or

$$\left|\log_2\alpha - \frac{p(n)}{2^{n+K}}\right| < \frac{1}{2^n}.$$

This implies that $\log_2\alpha$ is computable in time $O((B^{A_2})^{(\alpha^{2^K})^{2^n}})$. By Corollary 6.4, $\alpha$ is computable in time $O\big((B^{A_2})^{(\alpha^{2^K})^{2^{n+k}}}\big) = O\big((B^{A_2})^{(\alpha^{2^{K+k}})^{2^n}}\big)$ for $k \in \mathbb{N}$. □

**Corollary 6.7.** *If $\alpha\ (> 1) \in \mathbb{R}$ is not computable in time $C^{C^{2^n}}$ for any constant $C > 1$, then there is no finite rewriting system with derivational complexity equivalent to $n^\alpha$ or $\alpha^n$.*

About the derivational complexity of the forms $n^\alpha$ and $\alpha^n$, there is a gap between the necessity and the sufficiency in our results. We do not know whether $n^\alpha$ (or $\alpha^n$) is equivalent to the derivational complexity of a finite rewriting system for a real number $\alpha\ (> 1)$ computable in time $C^{C^{2^n}}$ for some $C > 1$ but not computable in time $C^{2^n}$ for any $C > 1$.

Also, there is a gap on the range of $\alpha$ for the function $n^\alpha$. We do not know if there is a finite rewriting system with derivational complexity equivalent to $n^\alpha$ for $1 < \alpha < 2$. In particular, is there a finite rewriting system with complexity equivalent to $n^{3/2}$?

## Acknowledgements

I thank Professor Friedrich Otto for valuable discussions during my stays in Kassel, Germany in 2009, 2010 and 2011. I also thank the anonymous referees for their helpful suggestions, which led to an improvement of the paper.

## References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, MA, 1974.
[2] J.-C. Birget, Infinite string rewriting systems and complexity, J. Symbolic Comp. 25 (1998) 759–793.
[3] R.V. Book, F. Otto, String-Rewriting Systems, Springer, New York, 1993.
[4] N. Dershowitz, Termination of rewriting, J. Symbolic Comp. 3 (1987) 69–115.
[5] T.V. Griffiths, Some remarks on derivations in general rewriting systems, Inf. and Control 12 (1968) 27–54.
[6] D. Hofbauer, Termination proofs by multiset path orderings imply primitive recursive derivation lengths, Theoret. Comput. Sci. 105 (1992) 129–140.
[7] D. Hofbauer, C. Lautermann, Termination proofs and the length of derivations, in: RTA1989, in: LNCS, vol. 355, 1989, pp. 167–177.
[8] J.E. Hopcroft, J.D. Ullman, Introduction to automata theory, in: Languages and Computation, Addison-Wesley, MA, 1979.
[9] G. Huet, D.S. Lankford, On the uniform holting problem for term rewriting systems, Raport Laboria 283, Institut de Recherche d'Informatique et d' AUtomatique, Le Chesnay, France, 1978.
[10] K. Madlener, F. Otto, Pseudo-natural algorithms for the word problem for finitely presented monoids and groups, J. Symbolic Comp. 1 (1985) 93–111.
[11] G. Moser, Proof theory at work: complexity analysis of term rewrite systems, Habilitation Thesis, Univ. Innsbruck, 2009.
[12] M.V. Sapir, J.-C. Birget, E. Rips, Isoperimetric and isodiametric functions of groups, Annals Math. 156 (2002) 345–466.
[13] K. Weihrauch, Computable Analysis, Springer, Berlin Heidelberg, 2000.